PCLIPS : Parallel CLIPS

Center for Productivity Enhancement
University of Massachusettes at Lowell

Coranth Gryphon Mark Miller

Introduction

PCLIPS (Parallel CLIPS) is a set of extensions to the CLIPS expert system language. CLIPS was developed by the Software Technology Branch of NASA, at Johnson Space Center.

PCLIPS is intended to provide an environment for the development of more complex, extensive expert systems. Multiple CLIPS expert systems are now capable of running simultaneously on separate processors, or separate machines, thus dramatically increasing the scope of solvable tasks within the expert systems.

An expert system such as CLIPS attributes its power to the flexibility inherent in its method of storing and disseminating information. An expert system is composed of a rule-base, or set of actions to be performed when certain conditions hold true, and a fact-base, that information the system has regarding the various objects and situations that it knows about. The facts, some of which the system has when it starts off and some of which it acquires through input, are filtered through chain of rules until either a dead-end is reached or a solution is achieved.

As a tool for parallel processing, PCLIPS allows for an expert system to add to its fact-base information generated by other expert systems, thus allowing systems to assist each other in solving a complex problem. This allows individual expert systems to be more compact and efficient, and thus run faster or on smaller machines.

PCLIPS is designed to be used as a tool for multi-processing, with each expert system being responsible for some portion of an overall project. This type of system would have each individual expert system module responsible for the execution of it own individual task, with interaction in the form of information exchange occurring only when necessary. With this model, all of the expert systems execute in parallel, rather than having a single expert system performing multiple roles. When a condition arises within one of expert system which one or more of other modules needs to know about, that information can simply be transferred, rather than requiring the original expert system to deal with the problem itself. This can save significant overhead, as well as making it much less likely that any given expert system will get saturated by having to deal with too much information at any one time.

Parallel Communications

PCLIPS is built upon a communications package specifically designed to handle communications between expert systems. The package operates on a familiar client-server-broker model, with the pclips expert system as the client, and is used to send facts between different pclips objects.

Upon startup, the pclips process spawns off another process, the *server*, whose job is to handle all incoming and outgoing communications with other pclips servers. When it is activated, this server 'registers' with a global broker, which resides upon a well known port of a well known machine. The combination of pclips expert system process and server process is known as a 'pclips object'.

Upon termination of the pclips (client) process, the server notifies the broker that it is now 'unregistering', and then terminates. If the server is terminated by some outside agent, it undergoes the same shutdown procedure, and then notifies the pclips (client) process which is free to deal with this problem according to its defined behavior. This behavior may be to terminate, or to spawn off another server, or to take any other defined behavior.

Names

Each pclips object is assigned a unique name which identfies it to other pclips objects. This name is communicated to the broker upon startup (registration), and the broker verifies that no other pclips object has chosen this name. If no name is specified at the command line, then the object generates one (of the form <*host*>-<*pid*>-<*port*>), based upon its current host, process id number, and the port that its server connects to.

Standalone pclips processes (those running without a server as an isolated expert system) use only the host and process id number when generating a name.

Classes

It is also possible to specify a class that the pclips object is an instance of. If no class is specifed, then the pclips object defaults to be a member of the "*pclips*" class. Classes are used to control access and to enable secure communications across certain zones.

A more complete class definitions system for pclips objects is under development which will allow for subclasses and superclasses, resulting in true object inheritance.

Zoned Communications

To prevent the excessive network traffic which would result if every pclips object always talked to every other pclips object, a zoning mechanism has been implemented. There are three actions that can be taken with regards to a zone.

First, the pclips object may 'post-enable' to a zone. The object sends a message to the broker, informing it of its intent. If the object is on the brokers list of those authorized to send a message over that zone (write), then the broker sends back to the server a list of all other pclips objects that can receive a message over that zone. Note that it is not necessary to be able to receive on a zone to send over that zone. When a pclips object is not going to send messages over a zone any more, it can 'post-disable' using the same procedure.

Second, the pclips object may send a message across a zone. Once 'post-enabled' to the zone, the pclips server maintains a list of all other objects that are to receive messages sent over that zone. The pclips (client) processes sends the message to its own server, which then traverses the list of those objects, connects to each one sequentially, and sends the message.

Finally, a pclips object can receive any messages sent on a zone. This is accomplished by sending a message to the broker informing it that the pclips object is 'subscribing' to the specified zone. The broker then sends back to the server the current list of those pclips objects which are allowed to post to that zone. The server then notifies each of those objects to update the internally maintained list of which objects to send to. A pclips object may at any time stop listening to messages over a particular zone by sending an 'unsubscribe-zone' message to the broker, which in turn notifies all pclips objects which are post-enabled to that zone.

Each pclips is automatically subscribed and post-enabled to the zone "*all*", subscribed and post-enabled to a zone based upon its host, "*local-<host>*", subscribed and post-enabled to a zone based upon its class, "*class-<class>*", post-enabled to a manager's zone, "*<class>-manager*", and subscribed to a personal zone based upon its name that only it may receive messages on. This personal zone, of the form "*personal-<name>*" allows point-to-point communications, thus facilitating the query-reply system described below.

In addition, other zones may be created during execution, subscribed or post-enabled to, and communicated over, all controlled by the expert system.

Under CLIPS 5.0, each zone an instance of the defclass "*Zone*", and each pclips object is considered an instance of the defclass "*PClips*". Individual pclips objects are kept track of by their instance name. Since point-to-point communications is done by sending to the zone "*personal-<name>*", the defclass "*PClips*" is a subclass of "*Zone*". This eliminates the conceptual distinction that would otherwise occur between personal zones, and group zones.

A more detailed zone definition mechanism is under developement which will allow zones to fit into the object-oriented paradigm, thus providing for inheritance by way of a zone heriarchy.

Industry Standards

The client-server object, with a global broker, conforms to the Object Management Group's model of an Object Management Architecture *[1]*. By utilizing this standard, greater internal consistency is assured, and more options are left open for other applications to utilize both the PCLIPS communications mechanism and to interface directly to a pclips object itself.

Enhancements to CLIPS

In addition to communications, other related extensions have been added to the basic CLIPS language. The ones discusses here are in some way related to the communications package described above.

Dynamic Construct Creation

Under current expert systems, anytime the user wishes to create a new rule, they must sit down with an editor, create that rule in a file, then load that rule into the expert system. If the system has a command line interface then they also have the option of stopping whatever processing they are doing, again typing in the entire rule by hand, and hopefully saving it out so they don't have to do it all again.

Using the tools provided here, rules (and for CLIPS 5.0, other constructs) can now be automatically generated from fact or instance definitions, thus reducing the work necessary to configure the expert system.

Construct Modification

A second facet of this mechanism is the ability to use existing constructs as a basis to generate new ones, and even to modify existing constructs by reading in the current form, editing it, and replacing the old construct witht he modified form. To this end, the ability to return and parse the pretty-print forms of constructs has been added, thus allowing a pclips expert system to treat these constructs as other forms of data.

Remote Construct Assertion

Using the communications mechanism described above, it is also possible to send directives, in the form of templated facts, to other pclips objects.These directives inform the receiving pclips expert systems in how to generate rules to deal with specific situations, or defines a new network-wide class of objects, or any other type of construct that can be auto-configured.

Constructs can also be stored in the archive system described above in a more compact form and generated, either locally or remotely, as needed.

Archive System

While the fact structure associated with any given fact takes a significant amount of storage space, there is even more that relates to that fact's effects on the rest of the expert system. The CLIPS manual *[2]* specifically mentions that "the fact-list should not be used as a data-base for storage of extraneous information." This statement holds even more for the instance list since there may be many outstanding instances that need to be kept track of for data-base considerations which should not show up under an instance query function search.

The solution to this dilemma is the archive system, which operates in alongside the normal fact and instance managers, and maintains a seperate list of information that the expert system can access, but which does not show up under normal fact or instance searches. Thus only those facts and instances which are needed for the execution of the expert system would be kept in the normal lists, while all others can be archived out.

Remote Archive

Also, using the communications package described above, it is possible to create a remote archive process, whose sole function is to maintain yet another seperate fact and instance list, this one being global to all of the pclips objects that can communicate with it. Rarely used information can be kept in this archive, to be made available on a query basis to any pclips object that needs it.

Construct Storage

An additional feature of both the local and remote archive systems is that, using the automated construct creation mechanism described below, it is possible to efficiently store out any construct form, thus freeing up much more memory in the process. Those rules, functions, class definitions, and message-handlers can then be loaded in only when needed. By storing them in a remote archive site, those constructs now become available to an entire network of pclips objects, on a need-to-know basis, without every pclips object being forced to maintain its own copy of each of them.

Other extensions

Direct Routers

The CLIPS router system allows great flexibility in terms of directing input and output, but it lacks the ability to use this mechanism internally, without resorting to file operations. The direct router system allows a pclips expert system to use internal string routers in much the same way that it uses file routers. The string routers "*direct*" and "*assert*" are automatically defined, the former for generic use, and the latter to provide a global fact construction mechanism. Other string routers can be created and initialized at run time, and then used by the pclips experts system as needed.

Environment Preservation

CLIPS inherently comes with the capability of saving out various portions of its current working state (facts, constructs, instances) to files. The environment preservation package makes use of that and automatically stores the environment when a pclips object terminates. This behavior is selectable, and controlled either at compile-time or at run-time.

This has three potential uses. The first is simply for debugging purposes. It can be very enlightening to look at the 'core' image of the pclips expert system. The second, and more practical use, is to allow a pclips object to 'simulate' any given event-sequence by saving out the current working state, then adding the simulated input, then restoring the environment afterwards.

Finally, the preservation mechanism can enable a pclips object to 'reincarnate' itself on the local host, or on a remote host that has access to the local file system. This is done by storing out the current environment, and then spawning off another copy of the pclips process, either locally or on a remote host, which would then load in the environment and continue with the knowledge of what had occurred. This can be done directly by the user, via the command line flag "*-restore*", or automatically if the appropriate flags are set.

System Operations

Currently, CLIPS allows access to operating system functions via the "*system*" function call. However, not only is this dependent upon specific operating systems, but it requires the expert system to keep track of what host type it is executing on, and to issue the commands appropriate to that system.

The functionality that has been added here is a standardization of the more commonly used command functions, and the ability for the pclips expert system to ignore the specifics of what operating system it is on when all it needs to do is simple file interactions.

In addition, the capability of communicating with users in the outside world has been incorporated into PCLIPS, both by providing a consistent interface to system specific "write/reply" utilities, and by enabling clips to send electronic mail.

There are going to be cases where the external action to be taken is independant, as far as the pclips expert system level, of the specific operating system. One example of this is the execution of a user application. To provide greater flexbility of control, the system interaction capabilities of CLIPS have been expanded to allow for creation of processes on either local or remote host, and to specify whether those processes are run directly by the pclips process or spawned off as child processes.

Finally, this allows for secure systems by eliminating the need for the entire pclips object to run as root when simple built-in system commands or simple stand-alone utilties (which themselves would have root access) can instead be spawned off and made to return the information to the pclips object via local communication with the server.

Host/Network Knowledge

Since PCLIPS makes use of network and host information, both for communication, and when executing processes on remote hosts, it was determined that the pclips expert system needed the ability to extract information about its local host, as well as remote hosts on connected subnets. This takes the form of utilities which allows the expert system to extract information about its host and its subnet environment from the machine itself, as well as being able to generate a 'subnet map' of the local networks which a host resides upon.

Command Line Arguments

Finally, much of the capability and flexibility of PCLIPS has been made readily usable by the addition of a large number of command line arguments.

These include the setting of all the internal flags used by the extensions, the ability to specify a name, class, or password for a pclips process, zone control (subscription and post-enabling) at startup, debuging behavior, both in terms of the clips "*watch*" command and for internel debugging and warning messages, as well as more flexibility to load in different formats of facts and contructs, whether to run as an isolated process, or expand to be a full pclips object (client and server pair), and finally the user can specify at run-time whether to drop into interactive or 'batch' mode.

Future Work

Through the use of parsing functions, information about the local host, its operating system, and its file system can be made directly available to a pclips expert system. This enables a pclips expert system to directly process externally generated information, without needing the user to translate.

PCLIPS Architecture

Certain source files of the original CLIPS code needed to be modified to provide the necessary access to the extensions listed above. The command line module now includes calls to read in pending messages from the server. The rule engine also now includes calls to the read in pending messages. Finally the "*clips-main*" module has been almost completely rewritten, allowing for a non-interactive execution mode different from the *RUN_TIME* setting provided for, as well as making the necessary calls to initialize other portions of the extended code.

In addition, there are many new user defined functions that a pclips expert system has access to. Some of these merely provide greater access to existing CLIPS mechanisms. Others provide system independant access to the underlying machine. In both cases, the expert system has been given as much flexibility of control as possible.

Finally, the entire communications package has been turned into a library, both to provide greater access, and to ensure ease of portability. This allows other programs to better interact with PCLIPS, and to make use of the already developed communications protocols.

Customization

Each group of extensions is fully controllable at the source code level through the setting of compilation flags. Original CLIPS uses a setup header include file to control its internal customization. PCLIPS uses an additional header file to control the extensions.

Other Programs

In addition to programs described above (pclips process, server, and broker), other utilities have been added to the PCLIPS family to enable greater flexibility and usage. Each of these makes use of the "*plib*" communication library described above.

Scratch-Pad

There is a scratch pad for communication with pclips objects, which is able to use either the zoned communications method described above, or to communicate directly with a specific pclips server on the local host. It also has a built in dribble file for debugging and log purposes.

Under a windowing environment (such as X-Windows, or the Apollo Domain pads), the scratch pad is designed to be run in a seperate window, thus allowing the pclips process to run non-interactively, and freeing up that window (if there is one) to show only expert system output and whatever debugging information is being displayed.

Pipe-Assert

To enable access to system utilities, a "*pipe-assert*" program exists which reads from standard input and transmits each line as fact (with appropriate prefacing) to the pclips server, which in turn returns it to the pclips process itself as a local fact. Other utilities have been written which use a built-in version of this to communicate directly to the server.

PCLIPS Security

Since PCLIPS can be used for sensitive projects (the MASE project *[3]*), it is important for PCLIPS to have internal security capabilities, to minimize the possibility of a foreign process reading messages it is not supposed to see, and writing out messages over zones that it is not cleared for. The first level of security in PCLIPS is achieved using the name/password scheme.

Passwords

A pclips process can be given a password, which allows for authentication of that pclips object. If a pclips object attempts to register with a "known" name, then that process must also supply the associated password for that object. Otherwise, the registration will fail. The password is sent as part of the registration message to the broker.

register name: <*name*> port: <*port*> pid: <*pid*> uid: <*uid*> host: <*host*>
ip: <*ip_address*> [password: <*password*>] \n

The password is currently sent in its unencrypted form. When the broker receives the "*register*" request, it encrypts the password, and then checks that encrypted form against against the entries in its password file.

Password/Access Control File

The broker maintains a password file, which contains the names and passwords of certain "known" pclips objects, as well as class information, and any restrictions or requirements on zone access.

It is possible to specify that certain names are reserved for certain users, or can only run on certain hosts, or that they can only belong to certain classes. If a pclips object which does not meet these criterion tried to use that name, it would fail to register.

Alternately, for a given specified class, it is possible to limit the allowable names that objects in that class can use. Likewise, for a given host, there can be restrictions on the names of objects that run on that host. A pclips object of a given name which attempts to belong to a restricted class, or run on a restricted host, would similarly fail to register.

Finally, zones can be limited such that only specific named objects are permitted to subscribe and/or post-enable to that zone.

Authorization Numbers

In order to maintain some form of security within PCLIPS, the broker now uses internally generated authorization numbers. Whenever an object registers, a unique authorization number, called the registration number, is passed back. This registration number is known only by the broker and the pclips object. Any further requests from the pclips object to the broker must include this registration number, as a way of authenticating future requests. This greatly reduces the potential of another process mimicking an already registered pclips object, using false requests to gain access to unauthorized information.

The unique authorization number is generated by the broker using the following algorithm: when the broker first starts up, it gets the current time from the system. This is then used a seed for a random number generator. Every time a new event occurs in the broker where a new authorization number is required, a call is made to the random number generator.

Zone Authorization

When an object subscribes to a zone, it is given a unique subscription authorization number for that zone. It is also given a list of all objects which are post-enabled to that zone, and each of their unique poster authorization numbers. The subscribing object then sends a message to each post-enabled object, which includes the unique poster authorization number for that object, and the subscription authorization number of the subscribing object. The post-enabled object then adds this subscription authorization number to its internally maintained list of objects subscribed to that zone. When this object later sends a message out over the zone, it will connect to a subscribed object, and include in the message the unique subscription authorization number for that receiving object. Thus, the subscriber can verify that the message it received actually came from a pclips object which was authorized to post on that zone by the broker.

Authorization numbers inhibit a foreign process from sending an illegitimate message to the pclips server, simply by writing out to that server's port. If the sending object has not gone through the authorization procedure, it should not have the necessary authorization number, and hence any incoming messages from it would be ignored. The authorization routine implemented by the broker attempts to verify that the process is a valid member of the PCLIPS family, and that the process is being run by a valid user.

Unique authorization numbers are needed for combination of object, zone, and access (post-enable or subscribe) to prevent legitimate pclips objects from getting enough information to by-pass the security of the broker. This is very important, since it is left up to the individual pclips objects to send messages directly to other objects, and to notify them of zone access changes, rather than relying upon a centralized message handler.

Specifically, if different subscribe and post-enable were not used, it would be possible for an object to get post-enable permission to zone, and then falsely inform other pclips objects that it is a subscribed to that zone, since it has the single authorization number for that zone, and thus gain illegal subscription access to that zone.

Notification Procedure

When an object post-enables to a zone, the broker sends it a list of all the pclips objects which are subscribed to that zone, along with their individual subscription authorization numbers. When the object posts (transmits a message) over that zone, it must include the subscription authorization number of the pclips object which is receiving that message.

Similarly, when an object subscribes to a zone, the broker sends it a list of all pclips objects which are post-enabled to that zone, along with each of their posting authorization numbers. It then notifies each of those objects that it is subscribed to this zone, and must include in that notification its own subscription authorization number. Those pclips objects each add the subscribed object, and its unique subscriber authorization number, to their internally maintained list of objects to send to.

Future Work

Under this current implementation, the security of PCLIPS is only as good as the security of the host running the broker. Since the broker runs as root on its host, an unauthorized user who gets access to root on the broker's machine would be able to read the broker's database (password and zone access control list) files, thus comprimising the entire authorization scheme. The fault-tolerant, multiple broker scheme currently under development eliminates this problem.

We also have the security hole inherent in unencrypted messages. If an unauthorized user was able to capture packets traversing the ethernet, the user would then be able to read passwords and authorization numbers. We will close this hole by developing an encryption capability, to be used when higher levels of security are required. This will have a performance trade-off, since outgoing messages would have to be encrypted at the point of origin, and then decrypted at the point of reception.

Expert System Applications

Given the above extensions, a number of new expert system mechanisms have already been added. Some of these are themselves extensions to PCLIPS, while others are simply examples of what can be done with those extensions.

The system described below, like most of the systems designed around these extenstions, is implemented using both templated facts and the CLIPS Object System. The reason for this is twofold; first, most of these mechanisms were origianally implemented under CLIPS 4.3, and thus the Object System did not exist. However, in many cases the object-oriented approach yielded greater functionality with lower overhead. Second, there are many users of CLIPS 5.0 who will not use either the rule-bases portion or the Object System of CLIPS, and providing both forms allows for the greatest possible distribution.

Query-Reply System

One example of the automated contruct configuration mechanism (described above) is the Query-Reply system which has been implemented, intended to allow differnt expert system to transfer information on a need-to-know basis. In other words, if an expert system running on one machine needed to know certain information which was known by an expert system running on a different machine, then the former would send a query to the latter, which would reply with the requested information.

A type of query, and the means of generating the correct reply to it, is specified. A rule is then automatically generated to take incoming queries and formulates the reply, which is then sent out to the specified reply point (usually the sender).

Templated Fact Implementation

A templated fact is specified which defines how the receiving expert system deals with a query.

```
(deftemplate AUTO-REPLY
        (field query-opcode (type WORD))
        (field auto-gen-report (allowed-words yes no))
        (field reply-point (allowed-words specified sender none))
        (field auto-post-enable (allowed-words yes no))
        (field retract-fact (allowed-words yes no))
        (field reply-salience (type NUMBER) (default 0))
        (multi-field query-funcs (type STRING)))
```

The "*query-opcode*" field specifies the type of query. The "*auto-gen-report*" field specifies whether the requested information should be printed out by the local expert system when a query is received.

The "*reply-point*" field is used to determine the point to send the information back to. The value "*specified*" means that the reply should be to the zone specified in the query message. A value of "*sender*" means reply to the sender of the query, regardless of the reply point specified in the message. A value of "*none*" means that no reply message should be sent. This is used in conjunction with report to display information for debugging purposes.

The "*auto-post-enable*" field determines if the replying pclips expert system should automatically attempt to post-enable to the reply-point zone.

The fields "*retract-fact*" and "*reply-salience*" are used when multiple replies are possible to a single query.

Finally, the "*query-funcs*" multi-field specify which functions (either primitive, deffunction, defgeneric, or message-handler) return the needed information. Each string is a single function call, with any needed arguments included.

When a query message is generated by the sending pclips expert system, it has a tag symbol added to it. This tag is used to uniquely associate a specific reply with a given query.

CLIPS Revisions

At the time of this writing, CLIPS 4.3 is supported to the limit of its functionality. There are many weaknesses found in the older version of CLIPS which have been fixed under later versions.

In those cases where our extensions have been mirrored, or superceded, by new functions found in CLIPS 5.0, our extensions have been eliminated, and the inherent CLIPS code used. All of the functionality that exists in our extensions CLIPS 4.3 are found under CLIPS 5.0, either inherent or added. In addition many other changes under the newer version have made possible extensions that could not easily be done, or were never thought of, in the older version of CLIPS. Some examples are listed below.

Because it was almost impossible to access fact addresses at the clips expert system level under CLIPS 4.3, the archive system dealt with fixed strings, tagged by a key name. This mechanism was completely revamped to take advantage of the greater accessability to fact addresses found under CLIPS 5.0, as well as expanding to include archival of instances.

The existance of the "*defglobal*" construct made the global variable system that was under developement obsolete, and thus it was dropped.

Since CLIPS 4.3 did not have the inherent capability to directly generate new constructs at runtime, a rule-creation mechanism was enginerred. This mechanism took strings and built the specified rule accordingly. This rule was then loaded in via a direct routing system. Later versions of CLIPS have this functionality inherent, using the "*build*" function call. Those parts of the old system which were still useful were instead moved up to the expert system level, using the "*deffunction*" construct.

With the addition of the CLIPS Object System, the entire communications package has been revamped to take advantage of it, while still retaining enough internal consistency that older versions of PCLIPS can still send and receive the same messages.

Conclusions

The basic approach throughout all of PCLIPS development has been to add primitive functionality to CLIPS, not simply more used-defined functions. By adding these primitive, the expert system can take advantage of features that before were inaccessible, or did not exist, and combine them as needed. Also, arbitrary distinctions between contruct and object types have been eliminated to provide parallel utlity to similar things wherever possible.

In addition to the work described is paper, there is an entire truth maintenance system that is being developed. It works in conjunction with the extensions described above, but is a seperate enough project to warrant its own paper *[4]*.

Future work includes both rounding out and solidifying the existing code, as well as adding even more major significant extensions in the areas of planning and predition, context dependant reasoning, goal-directed backward chaining, distributed agendas, and neural network interaction.

System Support

Written in native TCP/IP and using C, PCLIPS is portable to most multi-tasking platforms. Versions built on CLIPS 4.3 and 5.0 are currently supported under SunOS, System V Unix (Stellar and DG Aviion), DEC Ultrix (3.1 and 4.1), and Apollo Domain OS. A version running VAX VMS is currently under development

Bibliography

[1] Object Management Group Standards Manual, Draft 0.1 ; Richard M. Stoley, Ph.D.

OMG TC Document 90.5.4 ; (c) May 25, 1990

[2] CLIPS 5.0 Reference Manual, Volume 2 ; (c) January 11, 1991

Software Technology Branch, Lyndon B. Johnson Space Center, NASA

[3] MASE : Management and Security Expert; Mark Miller, Coranth Gryphon, et al.;

University of Massachusettes at Lowell, Center for Productivity Enhancement;

(c) August 1, 1991; Published at the Second Annual CLIPS Users Conference

[4] Temporal Reasoning Extensions to CLIPS; Coranth Gryphon, Marion Williams, et al.

Univesity of Massachusetes at Lowell, Center for Productivity Enhancement;

(c) August 15, 1991; a working paper